Story 1.8: Local Development Environment (Docker Compose + Observability)

Story ID: 1.8
Epic: E1 - Foundation & Infrastructure
Author: BMAD Development Team
Date: 2025-11-27
Status: ✅ COMPLETED

📋 Story Description

As a Developer
I want Docker Compose con todos los servicios para desarrollo local
So that Puedo correr el stack completo con `docker-compose up`

✅ Acceptance Criteria

1. Docker Compose Setup

Given Docker y Docker Compose instalados
When Ejecuto `docker-compose up -d` desde raíz del proyecto
Then Se levantan servicios:

- ✅ PostgreSQL 15 (puerto 5432)
- ✅ Kafka + Zookeeper (puertos 9092, 2181)
- ✅ Schema Registry (puerto 8081)
- ✅ HashiCorp Vault (puerto 8200)
- ✅ Prometheus (puerto 9090) - **NEW**
- ✅ Grafana (puerto 3000) - **NEW**

2. Health Checks

And Health checks pasan para todos los servicios

- ✅ PostgreSQL: `pg_isready -U siguser`
- ✅ Kafka: `kafka-broker-api-versions --bootstrap-server localhost:9092`
- ✅ Schema Registry: `curl http://localhost:8081/`
- ✅ Vault: `vault status`

- ✅ Prometheus: `curl http://localhost:9090/-/healthy`
- ✅ Grafana: `curl http://localhost:3000/api/health`

3. Spring Boot Integration

And La aplicación Spring Boot puede conectarse a todos los servicios

- ✅ Database connection pool healthy
- ✅ Kafka producer/consumer ready
- ✅ Vault secrets loaded
- ✅ Prometheus metrics exposed at `/actuator/prometheus`

4. Documentation

And README.md documenta:

- ✅ `docker-compose up -d` para iniciar
- ✅ `docker-compose down -v` para limpiar
- ✅ Ports mapping y URLs de acceso
- ✅ Credenciales default (solo dev)
- ✅ Health check verification scripts

📐 Implementation Details

Docker Compose Architecture

```
signature-router-network (bridge)
├── postgres:15-alpine
│   └── Port: 5432
│   └── Health: pg_isready
│   └── Volume: postgres-data
├── zookeeper (Confluent 7.5.0)
│   └── Port: 2181
├── kafka (Confluent 7.5.0)
│   └── Ports: 9092 (external), 29092 (internal)
│   └── Health: kafka-broker-api-versions
│   └── Depends: zookeeper
├── schema-registry (Confluent 7.5.0)
│   └── Port: 8081
│   └── Health: curl http://localhost:8081/
│   └── Depends: kafka
├── vault (HashiCorp 1.15)
│   └── Port: 8200
```

```
│   └── Health: vault status
│   └── Volume: ./vault/scripts
│   └── Dev Mode: VAULT_DEV_ROOT_TOKEN_ID=dev-token-123
├── prometheus (Prom 2.48.0) 🆕
│   └── Port: 9090
│   └── Health: curl /-/healthy
│   └── Volumes: ./observability/prometheus.yml, prometheus-data
│   └── Retention: 30 days
│   └── Scrapes: host.docker.internal:8080/actuator/prometheus
└── grafana (Grafana 10.2.0) 🆕
    └── Port: 3000
    └── Health: curl /api/health
    └── Volumes: grafana-data, ./observability/grafana/
    └── Provisioning: Datasources + Dashboards auto-loaded
    └── Credentials: admin/admin
```

Observability Stack (NEW in Story 1.8)

Prometheus Configuration

File: `observability/prometheus.yml`

Key Features:

- Scrape interval: 10s (faster than default for real-time monitoring)
- Target: `host.docker.internal:8080` (Spring Boot app on host)
- Metrics path: `/actuator/prometheus`
- Storage retention: 30 days
- Self-monitoring enabled

Scrape Configuration:

```yaml
scrape_configs:
  - job_name: 'signature-router-app'
    metrics_path: '/actuator/prometheus'
    scrape_interval: 10s
    static_configs:
      - targets: ['host.docker.internal:8080']
```

Grafana Configuration

Auto-Provisioning:

1. **Datasource** (`observability/grafana/provisioning/datasources/prometheus.yml`)
    - Automatically adds Prometheus as default datasource

    - No manual configuration needed
2. **Dashboards** (`observability/grafana/provisioning/dashboards/default.yml`)
    - Automatically loads dashboards from `observability/grafana/dashboards/`
    - Updates every 30s
    - Allows UI updates

Pre-Built Dashboard:

- **Signature Router - Overview** (`signature-router-overview.json`)
    - Application Status (UP/DOWN gauge)
    - HTTP Request Rate (req/s time series)
    - HTTP Latency Percentiles (P50, P95, P99)
    - JVM Memory Usage (Heap, Non-Heap)
    - Database Connection Pool (HikariCP)
    - CPU Usage (System, Process)

Health Check Verification Scripts

Windows PowerShell: `verify-health.ps1`

```
.\verify-health.ps1
```

Linux/Mac Bash: `verify-health.sh`

```
chmod +x verify-health.sh
./verify-health.sh
```

Features:

- ✅ Checks Docker container health status
- ✅ Verifies HTTP health endpoints
- ✅ Color-coded output (Green=Healthy, Red=Unhealthy, Yellow=Starting)
- ✅ Exit code 0 = all healthy, 1 = some unhealthy
- ✅ Troubleshooting tips on failure

📊 Metrics Exposed

Spring Boot Actuator Endpoints

Endpoint	Purpose	Public
`/actuator/health`	Health status of all components	✅
`/actuator/info`	Application info	✅
`/actuator/metrics`	Available metric names	❌
`/actuator/prometheus`	Prometheus-formatted metrics	❌

Key Metrics for Grafana

HTTP Metrics:

- `http_server_requests_seconds_count` - Total requests
- `http_server_requests_seconds_sum` - Total duration
- `http_server_requests_seconds_bucket` - Histogram buckets (for percentiles)

JVM Metrics:

- `jvm_memory_used_bytes{area="heap"}` - Heap memory
- `jvm_memory_max_bytes{area="heap"}` - Max heap
- `jvm_gc_pause_seconds_*` - GC pause times

Database Metrics:

- `hikaricp_connections_active` - Active DB connections
- `hikaricp_connections_idle` - Idle DB connections
- `hikaricp_connections_pending` - Pending connections

System Metrics:

- `system_cpu_usage` - System CPU (0-1)
- `process_cpu_usage` - Process CPU (0-1)

🎯 SLO Tracking

Banking-Grade SLOs (from PRD NFR-P1 to NFR-P10)

SLO	Target	Metric	Dashboard Panel
Availability	≥99.9%	`up{job="signature-router-app"}`	Application Status
P99 Latency	<300ms	`histogram_quantile(0.99, http_server_requests_seconds_bucket)`	HTTP Latency
Error Rate	<0.1%	`sum(rate(http_server_requests_seconds_count{status=~"5.."}[5m]))`	(Add custom panel)

🚀 Quick Start Guide

1. Start All Services

```
# From project root
docker-compose up -d

# Wait for all health checks to pass (30-60 seconds)
docker-compose logs -f
```

2. Verify Health

```
# Automated verification
.\verify-health.ps1   # Windows
./verify-health.sh    # Linux/Mac

# Manual verification
docker-compose ps
curl http://localhost:8200/v1/sys/health | jq .      # Vault
curl http://localhost:8081/ | jq .                   # Schema Registry
curl http://localhost:9090/-/healthy                 # Prometheus
curl http://localhost:3000/api/health | jq .         # Grafana
```

3. Start Spring Boot Application

```
.\mvnw spring-boot:run -Dspring.profiles.active=local

# Verify app health
curl http://localhost:8080/actuator/health | jq .
```

4. Access Observability Tools

Grafana:

1. Open http://localhost:3000
2. Login: `admin` / `admin`
3. Navigate: **Dashboards** → **Signature Router** → **Overview**

Prometheus:

1. Open http://localhost:9090
2. Navigate: **Status** → **Targets** (verify app is UP)
3. Navigate: **Graph** (query metrics manually)

5. Generate Load for Metrics

```
# Generate HTTP requests to populate dashboard
for i in {1..100}; do
  curl http://localhost:8080/actuator/health
  sleep 1
done
```

6. Stop Services

```
# Stop all services
docker-compose down

# Stop and remove volumes (clean state)
docker-compose down -v
```

🔧 Configuration Files

Docker Compose

File: `docker-compose.yml`

Volumes:

- `postgres-data` - PostgreSQL data persistence
- `prometheus-data` - Prometheus time-series storage
- `grafana-data` - Grafana dashboards and settings
- `./vault/scripts` - Vault initialization scripts
- `./observability/prometheus.yml` - Prometheus config

- `./observability/grafana/provisioning` - Grafana provisioning
- `./observability/grafana/dashboards` - Dashboard definitions

Networks:

- `signature-router-network` (bridge) - Isolated network for all services

Environment Variables

PostgreSQL:

- `POSTGRES_DB=signature_router`
- `POSTGRES_USER=siguser`
- `POSTGRES_PASSWORD=sigpass`

Kafka:

- `KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://localhost:9092`

Vault:

- `VAULT_DEV_ROOT_TOKEN_ID=dev-token-123` (⚠️ DEV ONLY)

Grafana:

- `GF_SECURITY_ADMIN_USER=admin`
- `GF_SECURITY_ADMIN_PASSWORD=admin`

🐛 Troubleshooting

Prometheus not scraping app

Symptom: Target DOWN in Prometheus UI

Solution:

```
# Verify app is running and exposing metrics
curl http://localhost:8080/actuator/prometheus

# Check Prometheus logs
docker-compose logs prometheus

# Verify host.docker.internal resolves
docker exec signature-router-prometheus ping -c 1 host.docker.internal
```

Grafana shows "No data"

Symptom: Dashboard panels are empty

Solution:

```
# 1. Verify Prometheus datasource
curl -u admin:admin http://localhost:3000/api/datasources | jq .

# 2. Test query in Prometheus
curl 'http://localhost:9090/api/v1/query?query=up' | jq .

# 3. Restart Grafana
docker-compose restart grafana
```

Vault health check failing

Symptom: `vault` container unhealthy

Solution:

```
# Initialize Vault secrets
docker-compose exec vault sh /vault/scripts/vault-init.sh

# Verify Vault status
docker-compose exec vault vault status
```

Port conflicts

Symptom: `ERROR: for postgres Cannot start service postgres: Ports are not available`

Solution:

```
# Check what's using the port
netstat -ano | findstr :5432  # Windows
lsof -i :5432                 # Linux/Mac

# Stop conflicting service or change port in docker-compose.yml
```

📚 Additional Documentation

- **Observability Stack README** - Detailed Prometheus/Grafana guide
- **README.md - Observability Section** - Quick reference
- **Architecture - Observability** - System design

✅ Testing & Verification

Acceptance Tests

Test 1: All services start successfully

```
docker-compose up -d
sleep 60  # Wait for health checks
docker-compose ps | grep -v "Up (healthy)"  # Should be empty
```

Test 2: Health checks pass

```
.\verify-health.ps1  # Should exit with code 0
```

Test 3: Spring Boot connects to all services

```
.\mvnw spring-boot:run -Dspring.profiles.active=local
curl http://localhost:8080/actuator/health | jq '.status'  # Should be "UP"
```

Test 4: Prometheus scrapes metrics

```
curl 'http://localhost:9090/api/v1/query?query=up{job="signature-router-app"}' |
jq '.data.result[0].value[1]'
# Should be "1" (UP)
```

Test 5: Grafana loads dashboard

```
curl -u admin:admin http://localhost:3000/api/dashboards/uid/signature-router-
overview | jq '.dashboard.title'
# Should be "Signature Router - Overview"
```

Manual Verification Checklist

- ✅ `docker-compose up -d` starts all 7 services
- ✅ All services reach `healthy` state within 60s
- ✅ PostgreSQL accepts connections on port 5432
- ✅ Kafka accepts connections on port 9092
- ✅ Schema Registry returns schemas on http://localhost:8081
- ✅ Vault API responds on http://localhost:8200
- ✅ Prometheus UI accessible on http://localhost:9090
- ✅ Grafana UI accessible on http://localhost:3000
- ✅ Spring Boot connects to all services

- [x] Prometheus scrapes Spring Boot metrics
- [x] Grafana dashboard displays metrics
- [x] `docker-compose down -v` stops and cleans up all services

📈 Definition of Done

- [x] Docker Compose file includes all 7 services (Postgres, Kafka, Zookeeper, Schema Registry, Vault, Prometheus, Grafana)
- [x] All services have health checks configured
- [x] Prometheus scrape configuration created
- [x] Grafana datasource auto-provisioning configured
- [x] Grafana dashboard created with 6 key metrics panels
- [x] Health check verification scripts created (PowerShell + Bash)
- [x] README.md updated with observability documentation
- [x] Observability stack README created
- [x] `.dockerignore` created for build optimization
- [x] All services pass health checks
- [x] Spring Boot successfully connects to all services
- [x] Documentation covers all acceptance criteria
- [x] Roadmap updated to mark Story 1.8 as complete

🎉 Story Completion

Status: ✅ COMPLETED
Completed Date: 2025-11-27

Deliverables:

1. ✅ `docker-compose.yml` - Updated with Prometheus + Grafana
2. ✅ `observability/prometheus.yml` - Prometheus scrape configuration
3. ✅ `observability/grafana/provisioning/` - Auto-provisioning configs
4. ✅ `observability/grafana/dashboards/signature-router-overview.json` - Pre-built dashboard
5. ✅ `observability/README.md` - Comprehensive observability guide
6. ✅ `verify-health.ps1` - Windows health check script

7. ✅ `verify-health.sh` - Linux/Mac health check script
8. ✅ `.dockerignore` - Docker build optimization
9. ✅ `README.md` - Updated with observability section
10. ✅ `docs/sprint-artifacts/1-8-local-development-environment-docker-compose.md` - This document

Next Steps:

- Ready to proceed with Epic 2: Signature Request Orchestration
- Story 2.1: Create Signature Request Use Case
- All infrastructure foundation (Epic 1) is now complete! 🎊

Author: BMAD Development Team
Last Updated: 2025-11-27
Version: 1.0